CoT-ICL Lab: A Synthetic Framework for Studying Chain-of-Thought Learning from In-Context Demonstrations

Vignesh Kothapalli[1,2], Hamed Firooz[2], Maziar Sanjabi[2]

[1]New York University, [2]LinkedIn AI,

Correspondence: vk2115@nyu.edu

Abstract

We introduce *CoT-ICL Lab*, a framework and methodology to generate synthetic tokenized datasets and systematically study chain-of-thought (CoT) in-context learning (ICL) in language models. *CoT-ICL Lab* allows fine grained control over the complexity of in-context examples by decoupling (1) the causal structure involved in chain token generation from (2) the underlying token processing functions. We train decoder-only transformers (up to 700M parameters) on these datasets and show that CoT accelerates the accuracy transition to higher values across model sizes. In particular, we find that model depth is crucial for leveraging CoT with limited in-context examples, while more examples help shallow models match deeper model performance. Additionally, limiting the diversity of token processing functions throughout training improves causal structure learning via ICL. We also interpret these transitions by analyzing transformer embeddings and attention maps. Overall, *CoT-ICL Lab* serves as a simple yet powerful testbed for theoretical and empirical insights into ICL and CoT in language models[1].

1 Introduction

Transformer-based language models (Vaswani et al., 2017) have demonstrated remarkable capabilities in tasks requiring emergent reasoning behaviors, such as few-shot ICL (Brown et al., 2020; Firooz et al., 2025) and CoT prompting (Wei et al., 2022; Nye et al., 2021; Kojima et al., 2022). *In-context learning* refers to a phenomenon wherein language models generalize to new tasks by conditioning on a small number of input-output examples without explicit parameter updates (Dong et al., 2022). Meanwhile, *chain-of-thought prompting* augments the input with explicit intermediate reasoning steps that can guide the model's generative process toward more accurate solutions (Wei et al., 2022). Despite the substantial performance gains witnessed in various natural language processing tasks (Kim et al., 2023), the precise mechanisms and architectural factors driving ICL and CoT remain only partially understood.

Recent studies have ventured into controlled synthetic tasks to understand how transformers learn in-context (Garg et al., 2022; Von Oswald et al., 2023; Bai et al., 2023). These works often rely on real-valued examples consisting of single-input and single-output pairs, and study if the transformers can learn linear or non-linear function classes. While these tasks facilitate theoretical analysis, they leave open the question of whether the findings *readily* extend to more complex or compositional settings, especially pertaining to discrete tokenized sequences. In a parallel line of research, investigations of CoT prompting for NLP tasks often rely on short, human-annotated explanations or heuristics, thereby limiting the variety and control of "reasoning" processes considered (Wang et al., 2022; Liu et al., 2024; Yang et al., 2024; Prabhakar et al., 2024). Although such strategies have yielded valuable insights, there does not exist a setup that unifies ICL and CoT and facilitates systematic probing of different aspects of complexity—ranging from vocabulary size and chain length (i.e, the number of tokens involved in the reasoning process) to the shape and sparsity of dependencies between tokens.

In this work, we introduce *CoT-ICL Lab*, a tokenized synthetic dataset generation framework, that is specifically designed for studying how transformer-based models acquire chain-of-thought reasoning in-context. Our framework differs from prior work in the following ways:

1. **Tokenized setup akin to language.** Unlike many purely numeric toy tasks, we consider inputs and chain tokens in a discrete token space (i.e, a custom vocabulary $\mathcal{V}$). This setup aligns closely with natural language prompting and fa-

[1]The code and documentation is available at: `https://github.com/kvignesh1420/cot-icl-lab`

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 14620–14642
July 27 - August 1, 2025

cilitates complexity control via vocabulary size.

2. **Decoupled structure and token processing functions.** We represent the causal structure of the 'reasoning' chain via a directed acyclic graph (DAG) and implement token processing via arbitrary MLP transformations of the corresponding 'unknown' data embeddings $\mathbf{E}_{\texttt{data}} \in \mathbb{R}^{|\mathcal{V}| \times d}$. This separation grants flexibility in controlling problem difficulty —e.g., by manipulating chain length, number of edges in the DAG, depth and activation functions in MLPs and the dimension of data embeddings d.

3. **Multi input-output ICL examples.** A majority of the efforts which study ICL in transformer models rely on (real-valued) single input-output examples in-context (Garg et al., 2022; Bai et al., 2023; Li et al., 2023b). Our setup addresses these limitations and allows researchers to use tokenized, multi-input multi-output examples in-context, which is closer to practice. To the best of our knowledge, this is the first work to introduce and analyze transformer models in such controlled settings.

4. **Ablation-friendly design.** By varying one component at a time (vocabulary, number of input or chain tokens per example, DAG connectivity, MLP complexity, or the underlying transformer architecture), researchers can precisely identify which facets of the problem most challenge the model's ICL and CoT capabilities.

Key Results. In addition to proposing the *CoT-ICL Lab* framework, we showcase how it can be used to gain insights into the abilities of decoder-only transformer models in ICL with and without CoT. Specifically:

- Transformer-models undergo phase transitions in accuracy while training on ICL problems. Such transitions are facilitated by model size, availability of more examples in-context and CoT prompting.

- We empirically show that the phase transition correlates with the alignment between model's token embeddings and the data/language embeddings $\mathbf{E}_{\texttt{data}}$. Furthermore, when utilizing a finite set of token processing functions to generate the *CoT-ICL Lab* data, this reduction in problem complexity facilitates the attention maps of the model to capture the underlying reasoning DAG and excel at ICL.

- In essence, we highlight an interplay between the problem complexity induced due to diverse token processing functions and the DAG structure. As DAG sparsity reduces and the number of token processing functions increases, we observed that larger models tend to adapt to such diversity in ICL problems and leverage CoT to outperform the smaller models. Thus, showcasing the intricacies involved in scaling the model size for ICL performance.

2 Related Work

In-Context Learning. Initially popularized by GPT-3 (Brown et al., 2020), in-context learning has garnered extensive attention for its surprising ability to generalize with just a few example prompts. Many investigations center on how transformers might implicitly perform gradient descent or implement other adaptation mechanisms in their hidden activations (Garg et al., 2022; Akyürek et al., 2024; Von Oswald et al., 2023; Bai et al., 2023). See (Dong et al., 2022; Zhou et al., 2024) for surveys on the topic. However, these analyses often assume real-valued examples and very simple data distributions, leaving room to explore richer compositional structures that can align with natural language tasks.

Chain-of-Thought. CoT prompting (Wei et al., 2022; Nye et al., 2021; Kojima et al., 2022; Chu et al., 2024) has emerged as an effective technique for eliciting more interpretable (and sometimes more accurate) intermediate reasoning from large language models. Despite empirical successes, debate persists as to whether models truly learn a generalized reasoning algorithm or simply latch onto superficial features (Wang et al., 2022). While some efforts (Liu et al., 2024; Prabhakar et al., 2024) systematically study CoT's potential, they often rely on limited or handcrafted tasks that do not fully capture the complexity of multi-step compositional processes.

Synthetic Tasks for Controlled Model Analysis. Synthetic tasks provide controlled environments that enable precise interventions, ablation studies, and theoretical insights into the model behavior and training dynamics (Garg et al., 2022; Von Oswald et al., 2023; Bai et al., 2023). However, existing synthetic settings generally remain numeric and follow overly restrictive Markovian assumptions (Edelman et al., 2024) (e.g., a single parent for each token). Our proposed *CoT-ICL Lab* extends these efforts by decoupling the causal structure from token-processing functions. We leverage directed acyclic graphs (DAGs) to control the branching factor in the chain generation, and MLPs for varied levels of token transformations. This design grants extensive configurability, encompassing vocabulary size, multi-input example length, chain length, DAG sparsity, MLP depth, activations and more.

Chain-of-Thought & Compositional Reasoning. Recent studies focus on dissecting CoT to assess how compositionality might emerge from ICL. For instance, (Li et al., 2023b) examines how CoT might be effectively disentangled into filtering and learning the intermediate features of the MLP components in the prompt. Furthermore, our work can be treated as a generalization of the `MechanisticProbe` approach by (Hou et al., 2023) where the filtering and reasoning tree construction process is not limited by the availability of natural language datasets. While these and related efforts (Yang et al., 2024; Prabhakar et al., 2024) represent significant progress toward understanding emergent reasoning, their experimental setups typically do not offer the systematic and fine-grained complexity that *CoT-ICL Lab* enables (see also Appendix F).

3 Preliminaries and Setup

Notation. Let $\{1, \cdots, K\} = [K]$. We consider a vocabulary $\mathcal{V}$ to represent the tokens of a synthetic language. Let $\mathcal{F}$ denote a class of functions that are compositional in nature. Formally, a function $f \in \mathcal{F}$ is composed of C sub-functions as: $f = f_C \circ f_{C-1} \cdots \circ f_1$. Given N input tokens $\mathbf{x} = (x_1, \cdots, x_N) \in \mathcal{V}^N$, the function f recursively generates C *chain tokens* $\mathbf{y} = (y_1, \cdots, y_C) \in \mathcal{V}^C$ as follows:

$$y_c = f_c(x_1, \cdots, x_N, y_1, \cdots, y_{c-1}), \ \forall c \in [C]. \quad (1)$$

Here $\mathbf{y}_{:C-1} = (y_1, \cdots, y_{C-1})$ are treated as the *intermediate tokens* and y_C as the *answer token*. The recursive process involves all the *input* and *intermediate tokens* to generate the *answer token* and presents a generalized setup to study CoT. The full notation list is presented in Table 3.

3.1 Compositional Nature of $\mathcal{F}$

Let $\mathcal{G}, \mathcal{H}$ denote the causal structure and token processing function classes respectively. In this work, we consider the sub-functions involved in the composition of $f = f_C \circ f_{C-1} \cdots \circ f_1$ to be formulated as $f_c = h_c \circ g_c$, where $g_c \in \mathcal{G}, h_c \in \mathcal{H}$. Given input tokens $(x_1, \cdots, x_N)$, the chain tokens in (1) are decomposed into:

$$\begin{aligned} y_c &= f_c\left(x_1, \cdots, x_n, y_1, \cdots, y_{c-1}\right) \\ &= h_c\left(g_c(x_1, \cdots, x_n, y_1, \cdots, y_{c-1})\right), \forall c \in [C] \end{aligned} \quad (2)$$

• **The causal structure function class $\mathcal{G}$.** This class represents functions which take an arbitrary number of tokens and filter a fixed number of $M \leq N \in \mathbb{N}$ tokens. These M parent tokens represent the causal dependency of a *chain token*.

• **The token processing function class $\mathcal{H}$.** These functions process the M selected tokens from $\mathcal{G}$ and output a single chain token. Thus, emulating an arbitrary ground-truth 'reasoning process' which the transformer models are trained to approximate.

3.2 Multi-Input ICL and CoT

Sequence design. We consider a generalized ICL problem of learning $f \in \mathcal{F}$ with (multi) input-output pairs in the token space. An *example* is defined as a vector of N input tokens and the corresponding *answer token*, as per (1). A collection of $K \in \mathbb{N}$ such *examples* results in a sequence $\mathbf{p}^K(f)$ as follows:

$$\mathbf{p}^K(f) = \left(x_1^{(i)}, \cdots, x_N^{(i)}, y_C^{(i)}\right)_{i=1}^{K}. \quad (3)$$

By including the *intermediate tokens* in an *example*, we obtain a *CoT example*, which is now a vector of N input tokens, and all the C *chain tokens*. The corresponding sequence $\mathbf{p}^K_{CoT}(f)$ is given as follows:

$$\mathbf{p}^K_{CoT}(f) = \left(x_1^{(i)}, \cdots, x_N^{(i)}, y_1^{(i)}, \cdots, y_C^{(i)}\right)_{i=1}^{K}. \quad (4)$$

4 *CoT-ICL Lab*: Data Generation

In this section, we present details about the synthetic data generation using *CoT-ICL Lab* and draw parallels with NLP tasks.

Language vocabulary embedding. To create synthetic training and evaluation datasets via the *CoT-ICL Lab* framework, we consider a vocabulary $\mathcal{V}$ of arbitrary size and associate with it a common data embedding matrix $\mathbf{E}_{\texttt{data}} \in \mathbb{R}^{|\mathcal{V}| \times d}$. Here d denotes the data embedding dimension and the entries are sampled i.i.d from $\mathcal{N}(0, 1)$. In particular, $\mathbf{E}_{\texttt{data}}$ will be leveraged by $h \in \mathcal{H}$ to process embeddings of the tokens and return a new token (see Figure 1).

Causal structure via DAGs. $\mathcal{G}$ is selected to be a class of topologically sorted DAGs whose (1) edge connectivity represents the causality involved in chain generation and (2) whose sparsity controls the usage of input and intermediate tokens. For notational simplicity, we represent DAGs in our setup as $\mathcal{G}(M, N, C)$. We sample one DAG per sequence and use it to create all (CoT-) examples within the sequence. For instance, given input tokens x_1, x_2, x_3, x_4 and chain tokens y_1, y_2, y_3, we illustrate in Figure 1 a DAG which maps $y_1 \leftarrow \{x_1, x_2\}$,

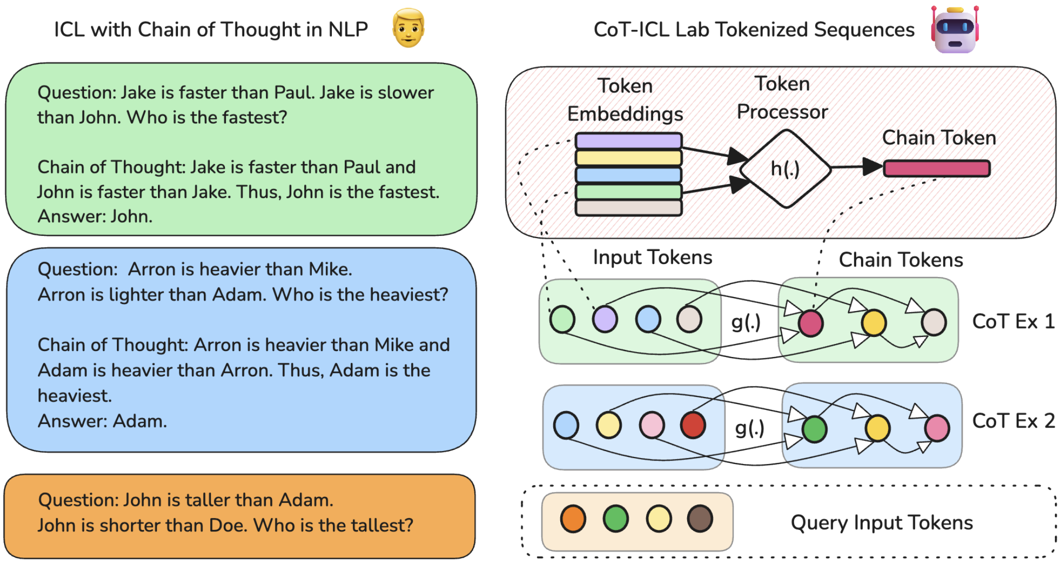


Figure 1: *CoT-ICL Lab* overview (right) and comparison to CoT ICL in NLP (left). The left figure illustrates 2 CoT examples (colored green and blue) presented in-context along with a question (colored in orange). A corresponding scenario using *CoT-ICL Lab* is presented on the right where we model the causal structure via the DAG $g \in \mathcal{G}$ and process the data embeddings $\mathbf{E}_{data}$ using the token processor function $h \in \mathcal{H}$.

$y_2 \leftarrow \{x_3, x_4\}$ and $y_3 \leftarrow \{y_1, y_2\}$. The possible structures that can be sampled using a particular choice of M, N, C controls the diversity of causal structures in our dataset of sequences.

Token processing via MLPs. The function class $\mathcal{H}$ is selected to be MLPs whose complexity is controlled by the activation ϕ such as `ReLU`, `SiLU`, `LeakyReLU`, `Identity`, and the depth $l \in \{1, 2, 3, 4, 5\}$. To generate a single chain token, we randomly initialize an MLP based on l, ϕ and use it to process the embeddings of the M parent tokens. We take the mean of the M final layer features, apply the activation function again and multiply with $\mathbf{E}_{\texttt{data}}^\top$ to obtain a chain token via arg-max. For notational simplicity, we represent MLPs of depth l and activation ϕ as $\mathcal{H}(l, \phi)$ (see algorithm in Appendix A.1). Thus, we sample C MLPs per sequence (one for each chain token) and use them to generate the chain tokens of all K examples within the sequence. In essence, these token processing functions are shared across the ICL examples in each sequence but differ across sequences. We present comprehensive details about the (1) distribution of tokens, and (2) the flexibility of our setup in terms of simulating the complexity of real world datasets in Appendix A.

5 Model Training and Evaluation

Training. We employ NLP style next-token prediction training of decoder only transformer (`TF`) models (Radford et al.) with Cross-Entropy (`CE`) loss. We employ the supervised fine-tuning strategy to compute the `CE` loss only on the K answer tokens for $\mathbf{p}^K$ and on all the $K \times C$ chain tokens for $\mathbf{p}^K_{CoT}$ (Garg et al., 2022; Bai et al., 2023; Li et al., 2023b).

Evaluation via `accuracy`. To measure the ICL ability of a `TF` model, we measure the `accuracy` of predicting the answer token of the query (final) example. Formally, we generate the evaluation sequence prefix with $K-1$ in-context examples and append the query input tokens $\tilde{\mathbf{x}} = (\tilde{x}_1, \cdots, \tilde{x}_N) \in \mathcal{V}^N$ at the end. For the query input $\tilde{\mathbf{x}}$, the ground truth chain tokens $\tilde{\mathbf{y}} = (\tilde{y}_1, \cdots, \tilde{y}_C) \in \mathcal{V}^C$ are generated by the recursive formulation given in (1) using a test function $\tilde{f} \in \mathcal{F}$. Note that there is a difference in model predictions w/ and w/o CoT as follows:

$$\begin{aligned} \hat{y}_{pred} &:= \texttt{TF}\left(\mathbf{p}^{K-1}(\tilde{f}), \tilde{\mathbf{x}}\right) \quad & w/o\ CoT, \\ \hat{y}_{pred} &:= \texttt{TF}^{\circ C}\left(\mathbf{p}^{K-1}_{CoT}(\tilde{f}), \tilde{\mathbf{x}}\right) \quad & w/\ CoT. \end{aligned} \tag{5}$$

Here $\texttt{TF}^{\circ C}(\cdot)$ represents the C-step auto-regressive greedy token generation by the model (without any teacher-forcing) as follows (Li et al., 2023b):

$$\hat{y}_C = \texttt{TF}\left(\mathbf{p}^{K-1}_{CoT}(\tilde{f}), \tilde{\mathbf{x}}, \underbrace{\hat{y}_1, \cdots, \hat{y}_{C-1}}_{previous\ step\ outputs}\right). \tag{6}$$

Intuitively, when using CoT sequences, we allow the model to generate $C-1$ intermediate tokens, followed by the final token $\hat{y}_{pred} = \hat{y}_C$. Given an evaluation dataset of $\widetilde{T}$ sequences, the `accuracy` is formulated as $\texttt{accuracy} = \frac{1}{\widetilde{T}} \sum_{t=1}^{\widetilde{T}} \mathbb{I}_{\hat{y}_{pred}=\tilde{y}_C}$.

On intermediate tokens. Since we have access to all the ground truth chain tokens using $\tilde{f}$, we measure the `accuracy` of predicting them based on $\hat{y}_c, \forall c \in [C]$. In fact, in Section 6.2, we show a gradual error propagation phenomenon which results in higher `accuracy` values on tokens at the beginning of the chain and lower `accuracy` at the end.

Models. We create three models `TF-4, TF-8, TF-12` with varying depth based on the Llama-3 architecture (Dubey et al., 2024) for our experiments (see Table 1). We ensure that depth is the only varying design factor in the architecture to facilitate a systematic study of the model performance.

Learning the 'unknown' embeddings $\mathbf{E}_{\texttt{data}}$. Recall that the fixed embedding matrix $\mathbf{E}_{\texttt{data}} \in \mathbb{R}^{|\mathcal{V}| \times d}$ that was used to generate the training sequences is unknown to the `TF` models. To understand the effect of training on the learnable embeddings $\mathbf{E}_{\texttt{TF}}$ of the `TF` models, we measure the subspace similarity between the left singular bases of $\mathbf{E}_{\texttt{data}}$ and $\mathbf{E}_{\texttt{TF}}$ (Zhu and Knyazev, 2013). Let $d < |\mathcal{V}|$ and denote SVD of $\mathbf{E}_{\texttt{data}}$ and $\mathbf{E}_{\texttt{TF}}$ as follows to obtain:

$$\mathbf{U}_{\texttt{data}}\mathbf{S}_{\texttt{data}}\mathbf{V}_{\texttt{data}}^\top = \mathbf{E}_{\texttt{data}}; \quad \mathbf{U}_{\texttt{TF}}\mathbf{S}_{\texttt{TF}}\mathbf{V}_{\texttt{TF}}^\top = \mathbf{E}_{\texttt{TF}}$$
$$\texttt{sim}(\mathbf{E}_{\texttt{data}}, \mathbf{E}_{\texttt{TF}}) := \frac{1}{d} \cdot \left\| \mathbf{U}_{\texttt{data}}^\top[:d]\mathbf{U}_{\texttt{TF}}[:d] \right\|_F. \quad (7)$$

Here $\texttt{sim}(\cdot,\cdot)$ allows us to measure how well the subspaces of $\mathbf{E}_{\texttt{data}}, \mathbf{E}_{\texttt{TF}}$ are aligned, and a higher value indicates that the `TF` model is learning the token space of the target language embeddings.

Experimental setup. To present our empirical findings based on training `TF` models on *CoT-ICL Lab*, we use the following common parameters to create our datasets. We create a training dataset of size $T = 32 \times 10^5$, evaluation dataset of size $\widetilde{T} = 10^4$ and use $d = 10$ along with $\mathcal{H}(1, \texttt{LeakyReLU})$. In particular, as also mentioned in Section 4, we do not put limitations on the cardinality of $\mathcal{G}, \mathcal{H}$. See Appendix B for details on the hardware, training resources and hyper-parameters used for experiments.

6 Results

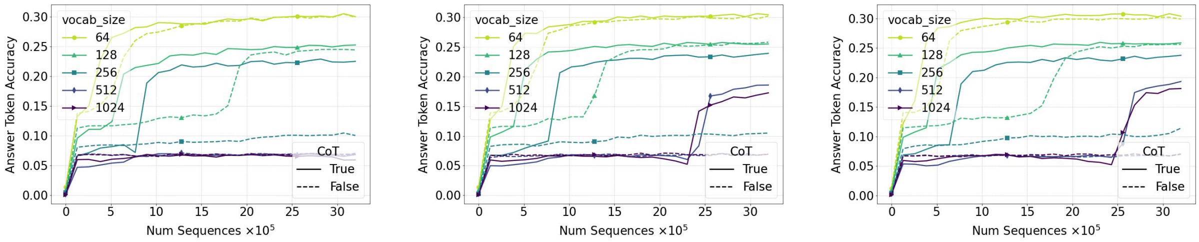
6.1 Effect of Vocabulary Size $|\mathcal{V}|$

As shown in Figure 1, our setup aims to mimic ICL and CoT problems in NLP. However, our synthetic language setup does not have the same priors and rules as tokens in natural language. In this section we test if such a problem is learnable to non-trivial levels of accuracy by `TF` models that only observe patterns in the ICL examples. We vary the size of the vocabulary as per $|\mathcal{V}| = \{64, 128, 256, 512, 1024\}$ along with $N = 4, M = 4, C = 2, K = \{30, 40\}$ and show that the `TF` models achieve non-trivial performance. To the best of our knowledge no prior work has done experiments with synthetic datasets of this vocabulary and model size (see Table 2).

Smaller models fail to leverage CoT for ICL with larger vocabularies. When $K = 30$, observe from Figure 2 that towards the end of training, the evaluation `accuracy` for $|\mathcal{V}| = \{64, 128\}$, is almost the same for CoT and non-CoT cases across all the three models. However, notice that CoT based ICL results in higher evaluation `accuracy` after training on relatively less number of sequences than the non-CoT approach, i.e **CoT results in faster transitions in performance**. More importantly, the benefits of CoT are prominent in `TF-4` when $|\mathcal{V}| = 256$ (see Figure 2a). In this case, CoT based ICL results in a sudden jump in `accuracy` after training on $\approx 8 \times 10^5$ sequences, but the non-CoT approach fails to exhibit such behavior even towards the end of training. The same observation can be made for `TF-8` (Figure 2b), `TF-12` (Figure 2c). Nonetheless, the benefits of model depth are evident when $|\mathcal{V}| = \{512, 1024\}$ as `TF-4` fails to leverage CoT with such large vocabularies whereas `TF-8`, `TF-12` clearly show the jumps towards higher `accuracy` (see Figure 2a vs Figure 2b, Figure 2c).

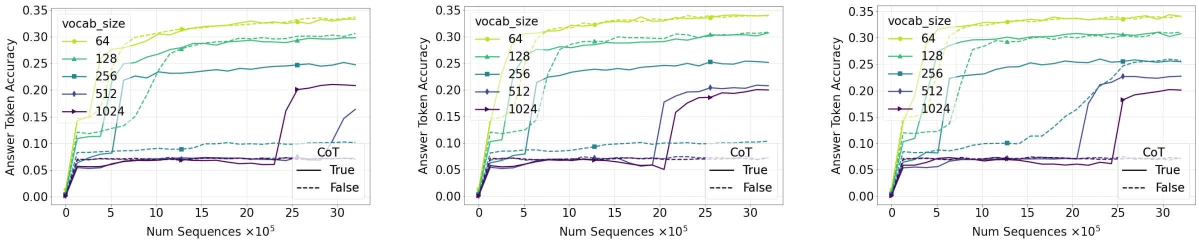
More ICL examples facilitate smaller models to leverage CoT. By increasing K to 40, we make an interesting observation that even a smaller `TF-4` model can perform on-par with larger `TF-8`, `TF-12` models in the difficult setting of $|\mathcal{V}| = \{512, 1024\}$ (see Figure 3). However, the role of model size comes into play when considering standard non-CoT examples where `TF-12` gradually improves with $|\mathcal{V}| = 256$ while `TF-4`, `TF-8` show a saturated curve (see blue dotted line in Figure 3c vs Figure 3a, Figure 3b). See Appendix C.3 for experiments with smaller K and Appendix C.4 for experiments with training on $3\times$ larger datasets, which highlight the role of model size for achieving better `accuracy`.

Embedding subspace similarity correlates with transitions in evaluation `accuracy`. Recall that we employ a common $\mathbf{E}_{\texttt{data}}$ to prepare our training and evaluation sequences. We noticed that $\texttt{sim}(\mathbf{E}_{\texttt{data}}, \mathbf{E}_{\texttt{TF}})$ serves as a useful metric to potentially explain the `TF-4` models earlier transition to



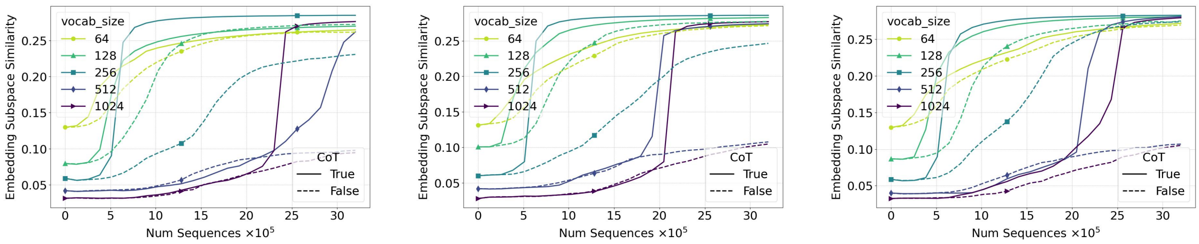
(a) `TF-4` (b) `TF-8` (c) `TF-12`

Figure 2: `accuracy` by varying $\mathcal{V}$ with $\mathcal{G}(M = 4, N = 4, C = 2), \mathcal{H}(1, \texttt{LeakyRelu}), d = 10, K = 30$.



(a) `TF-4` (b) `TF-8` (c) `TF-12`

Figure 3: `accuracy` by varying $\mathcal{V}$ with $\mathcal{G}(M = 4, N = 4, C = 2), \mathcal{H}(1, \texttt{LeakyRelu}), d = 10, K = 40$.



(a) `TF-4` (b) `TF-8` (c) `TF-12`

Figure 4: $\texttt{sim}(\mathbf{E}_{\texttt{data}}, \mathbf{E}_{\texttt{TF}})$ by varying $\mathcal{V}$ with $\mathcal{G}(M = 4, N = 4, C = 2), \mathcal{H}(1, \texttt{LeakyRelu}), d = 10, K = 40$.

a higer evaluation `accuracy` with $|\mathcal{V}| = 1024$, when compared to $|\mathcal{V}| = 512$ in Figure 3a. From Figure 4a, notice that $\texttt{sim}(\mathbf{E}_{\texttt{data}}, \mathbf{E}_{\texttt{TF}})$ transitions to a higher value after $\approx 23 \times 10^5$ steps, which exactly coincides with the transition point in Figure 3a. Similarly, the delay in alignment with $|\mathcal{V}| = 512$ in Figure 4a is indicative of a delay in the evaluation `accuracy` transition in Figure 3a. This is an interesting and unique aspect of *CoT-ICL Lab* that allows us to interpret and understand how the models learn relationships between the concepts in the synthetic language.

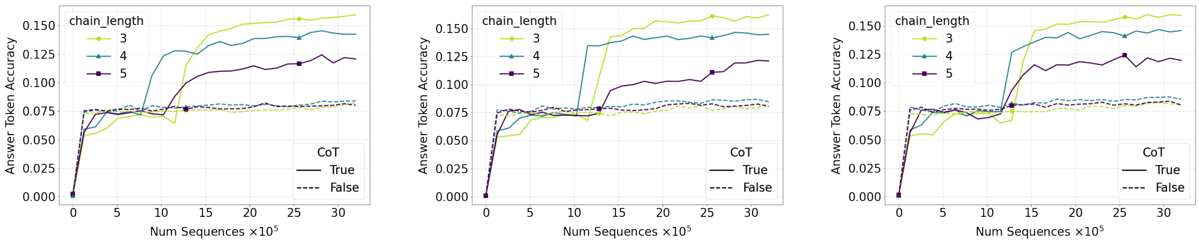
6.2 Effect of Chain Length (C)

The number of intermediate steps involved in the reasoning process is typically indicative of the complexity of NLP tasks. Its counterpart in our *CoT-ICL Lab* is the chain length C. By choosing $|\mathcal{V}| = 1024, N = 4, M = 4$ and varying $C = \{3, 4, 5\}$, we examine and show that longer chains result in harder problems that the models without CoT would not be able to solve effectively (Figure 5).

Longer chains result in lower `accuracy` across all model sizes. As the chain length increases, we can observe from Figure 5 that the evaluation `accuracy` is consistently lower for all models. Since the number of possibilities for the parents of the last chain token increases with C, we hypothesize that the models fail to adapt to such increased difficulty. This is indeed what we observe in Figure 6 where the `accuracy` consistently goes down for tokens towards the end of the chain across all model sizes.

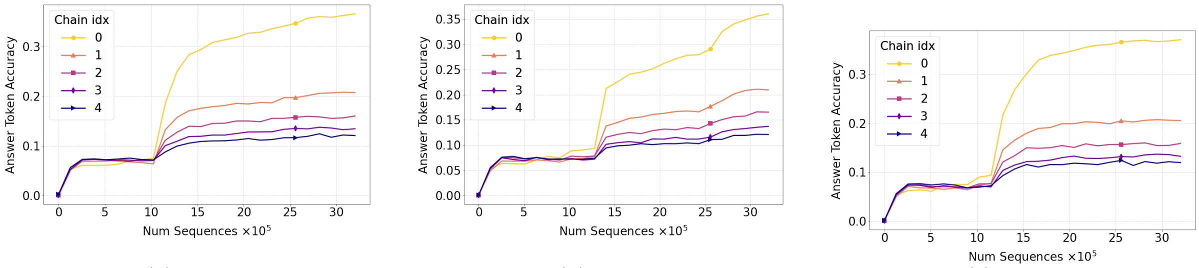
6.3 Effect of Number of Parent Tokens (M)

The dependency between tokens in NLP tasks is a variable that is usually not controllable (see Fig-



(a) `TF-4` (b) `TF-8` (c) `TF-12`

Figure 5: `accuracy` by varying C with $\mathcal{G}(N=4, M=4), \mathcal{H}(1, \texttt{LeakyRelu}), d=10, |\mathcal{V}|=1024, K=40$.



(a) `TF-4` (b) `TF-8` (c) `TF-12`

Figure 6: `accuracy` of predicting all the *chain tokens* (of the query *CoT example* in the evaluation sequences) with $\mathcal{G}(N=4, M=4, C=5), \mathcal{H}(1, \texttt{LeakyRelu}), d=10, |\mathcal{V}|=1024, K=40$.

ure 1). But in *CoT-ICL Lab* it can be controlled via M. By choosing $|\mathcal{V}|=1024, N=4, C=4$ and varying $M=\{1,2,3\}$, we observe that larger M (i.e dependency on more prior tokens) can make the problem harder for the models (see Figure 7).

Large models outperform small ones when DAGs are sparse. When $M=1$, we make an interesting observation that `TF-8` (Figure 7b), `TF-12` (Figure 7c) models exploit CoT towards the end of training to significantly outperform the `TF-4` model (Figure 7a). This shows an interesting difference in the training dynamics of deeper models, which could be an interesting research question for future works.

6.4 Ablations with $\mathcal{G}$ and $\mathcal{H}$

Our *CoT-ICL Lab* datasets, in their most general form, i.e. no restrictions on the cardinality of $\mathcal{G}$ and $\mathcal{H}$, are quite difficult for the models to solve (based on the low evaluation `accuracy` observed in prior sections). Owing to the flexibility of our setup, we decouple the effect of $\mathcal{G}$ and $\mathcal{H}$ to better understand the source of difficulty for the ICL problems.

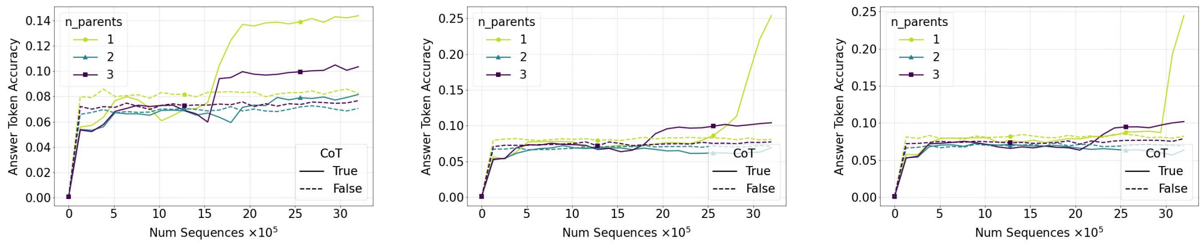
• **Fixed DAG structure.** We follow the same setup as Section 6.3 and sample different token processing MLPs but choose a fixed random DAG for all the training and validation sequences.

• **Fixed token processors.** Contrary to the above case, we sample random DAGs for different sequences but choose C fixed MLPs, one per chain location, as our token processors for all sequences.

By comparing Figure 8a, for fixed DAG, and Figure 8b for fixed token processors, we can observe that the **models reach higher accuracies in the fixed token processor setting**. This points to the possibility that the dense transformer models we consider here can identify the causal structure of the chain generation process with ease. We verify this by analyzing the attention map of the models trained in the fixed token processor setting (explained below). Furthermore, this gives us another lever to adjust the difficulty of synthetic datasets in *CoT-ICL Lab*, i.e adjusting the number of possible token processors (from infinite in the general case to a finite set). See Appendix C.5 for experiments with `TF-8` and `TF-12`.

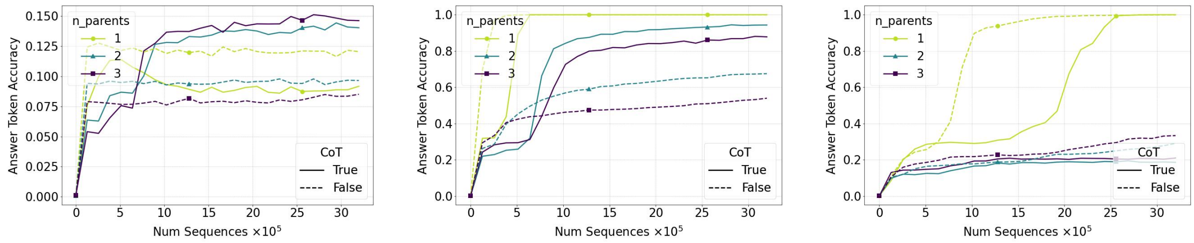
6.4.1 Attention Maps reflect the DAG

We consider models trained in the fixed token processor setting and analyze the attention map $\mathbf{A}$ averaged across all the heads $\mathbf{A}_h, h \in [H]$ of the last layer. Formally $\mathbf{A} := \frac{1}{H}\sum_{h=1}^{H} \mathbf{A}_h$. We plot such $\mathbf{A}$ for the last two examples of a single validation sequence in Figure 9a for $M=1$. Notice that most of the attention scores are almost zero and the ones with large values correspond to the parent tokens



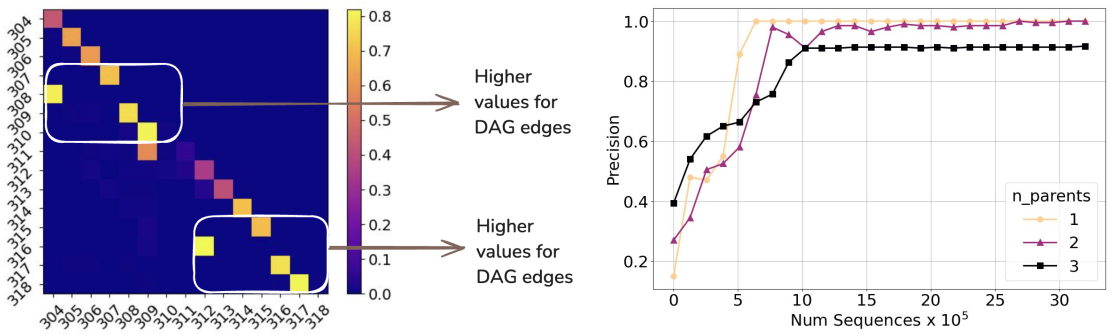
(a) `TF-4` (b) `TF-8` (c) `TF-12`

Figure 7: `accuracy` by varying M with $\mathcal{G}(N = 4, C = 4), \mathcal{H}(1, \texttt{LeakyReLU}), d = 10, |\mathcal{V}| = 1024, K = 40$.



(a) `TF-4` (Fixed DAG) (b) `TF-4` (Fixed 4 MLPs) (c) `TF-4` (Fixed 40 MLPs)

Figure 8: Ablation experiments with fixed DAG (a) and Fixed MLPs (b, c) for the `TF-4` model by measuring `accuracy` with varying M and $\mathcal{G}(N = 4, C = 4), \mathcal{H}(1, \texttt{LeakyRelu}), d = 10, |\mathcal{V}| = 1024, K = 40$.



(a) Average attention map **A** (b) `Precision` during training

Figure 9: (Left) Mean attention matrix (last 16 rows and columns) of all heads for the final layer of a trained `TF-4` model. The highlighted regions show that the model is attending to the correct parent tokens $y_1 \leftarrow \{x_4\}, y_2 \leftarrow \{x_1\}, y_3 \leftarrow \{y_1\}, y_4 \leftarrow \{y_2\}$ to generate the chain tokens. (Right) `Precision` in detecting the parent tokens using average attention map **A** of a trained `TF-4` model.

of each chain token, i.e. it matches the underlying DAG structure for this sequence. See Appendix D for illustrations with $M = 2, 3$.

Quantitative measurement via `Precision`. To quantify the attention map analysis to diverse settings, we calculate the `Precision` of identifying the parent tokens which are needed to generate the final answer token in the evaluation sequences. Let us assume that G is the set of ground truth parent tokens to formulate `Precision` as follows:

$$\begin{aligned} \mathbf{A}_{\text{query}} &:= \mathbf{A}[-1, -(N + C - 1) :] \\ \tilde{G} &:= \texttt{argsort}(\mathbf{A}_{\text{query}})[-M :] \\ \texttt{Precision} &:= \frac{\sum_{i \in \tilde{G}} \mathbb{I}(i \in G)}{M}. \end{aligned} \tag{8}$$

Figure 9b shows that as the model accuracy rises in this setting, the `Precision` of detecting the right DAG structure increases as well. It is worth noting that we repeated such an analysis for datasets with infinitely possible token processors but could not find such patterns. We believe that formalizing advanced tools for understanding how/if models detect causal structures is an important research area.

6.4.2 On Finite Token Processors

So far, we have considered the extreme cases of sampling from infinite token processors for every sequence or sample once and keep them fixed for all the sequences. Next, we show that by controlling the diversity of sequences by using a fixed collection (40 in this case, i.e 10 tuples of $C = 4$ MLPs.) of token processors to choose from and generate the sequences, the `TF` models require relatively more training sequences, compared to the fixed token processor setting above, to transition to higher accuracies (see Figure 8c). Intuitively, the final accuracy of the model for this setting is higher than the infinite token processor case (Figure 7a) and lower than the fixed token processor case (Figure 8b).

7 Connection to NLP

Tokenized prompts associated with NLP tasks are grounded in real-world knowledge. But due to the unknown causal structure of the underlying data it is hard to do controlled experiments with these datasets. On the other hand, although the sequences in *CoT-ICL Lab* are not associated with any grounded semantic information, we observed that models that are pre-trained on NLP learn *CoT-ICL Lab* tasks better and faster. Moreover, when examining the NLP models' attention maps, while performing reasoning on math tasks, we see sparse patterns in attentions between tokens (Appendix E). This motivates the design choice of using sparse causal structures in *CoT-ICL Lab*.

7.1 Training NLP Pre-Trained Models on *CoT-ICL Lab*

Setup. We consider the open-source pre-trained `Llama-3.2-1B-Instruct` model (from HuggingFace) and its random weight counterpart. The `model.resize_token_embeddings()` API is used to restrict the vocabulary size to 1024 for both models. The models are then trained on our synthetic *CoT-ICL Lab* dataset using the same experimental setup as in Section 6.1 with $\mathcal{G}(M = 4, N = 4, C = 2), \mathcal{H}(1, \texttt{LeakyRelu}), d = 10, K = 40$ and $|\mathcal{V}| = 1024$. Note that the vocabulary $\mathcal{V}$ and the data embeddings $\mathbf{E}_{\texttt{data}}$ used in the *CoT-ICL Lab* does not have any connection to the initial embeddings of the pre-trained model.

NLP pre-trained models transition faster than random counterparts. As shown in Figure 10 the pre-trained model transitions to a higher `accuracy` in the early phases of training with CoT sequences. On the contrary the random model exhibits such a transition after being trained on $\approx 13 \times 10^5$ sequences with CoT. Towards the end of training, the pre-trained model has an overall `accuracy` of 0.25, compared to 0.22 for the randomly initialized model. In the non-CoT training case, the pre-trained model has a final `accuracy` of 0.25 and clearly outperforms the random counterpart (which has an `accuracy` of 0.08). Note that the randomly initialized model does not exhibit a transition throughout training with non-CoT sequences. These results highlight that the pre-trained model has both higher accuracy w/ and w/o CoT and exhibits faster transition/improvement during training on *CoT-ICL Lab* data (with the difference being significant when CoT is disabled). Thus, hinting at a much deeper and interesting connection/similarity between the patterns that the Llama model learnt from natural language and the ones it is being trained and evaluated on in the synthetic setup.

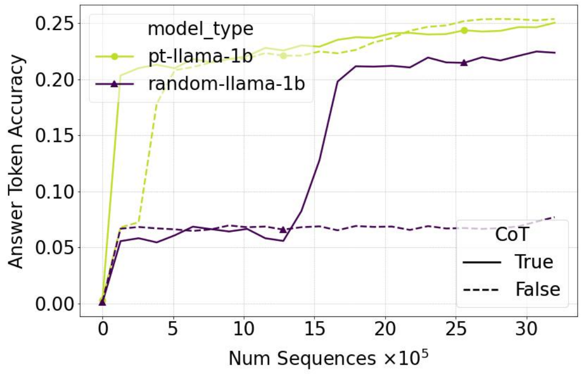


Figure 10: `accuracy` of pre-trained and random init `Llama-3.2-1B-Instruct` on *CoT-ICL Lab* .

8 Conclusion

In this paper, we introduced *CoT-ICL Lab*, a framework to generate synthetic Chain-of-Thought sequences and systematically study the role of CoT for In-Context Learning tasks. We used the flexibility and controllability of this framework to (1) generate synthetic multi input-output ICL datasets, (2) design controlled experiments to gain better insights into the role of CoT for ICL and (3) interpret the model behavior via embeddings and attention map analysis. We believe these insights, and many more that could be extracted by experimenting with *CoT-ICL Lab*, would play a crucial role in better understanding CoT for ICL in NLP, which is of utmost importance for the success of large language models.

9 Limitations

While *CoT-ICL Lab* is designed to closely mirror the chain-of-thought process in in-context learning for NLP, we acknowledge that its synthetic nature does not fully capture the linguistic properties of natural language. Specifically, *CoT-ICL Lab* tokens are not grounded in real-world concepts and therefore do not inherently align with the priors that govern natural language tokens. Consequently, researchers utilizing *CoT-ICL Lab* for experimentation should carefully weigh its advantages—such as flexibility and controllability—against its limitations, particularly its synthetic nature, and consider the potential impact on their results.

Acknowledgments

The authors would like to thank the anonymous reviewers for providing constructive feedback and improving our work.

References

Kwangjun Ahn, Xiang Cheng, Hadi Daneshmand, and Suvrit Sra. 2023. Transformers learn to implement preconditioned gradient descent for in-context learning. *Advances in Neural Information Processing Systems*, 36:45614–45650.

Ekin Akyürek, Bailin Wang, Yoon Kim, and Jacob Andreas. 2024. In-context language learning: Architectures and algorithms. In *Forty-first International Conference on Machine Learning*.

Jeff Alstott, Ed Bullmore, and Dietmar Plenz. 2014. powerlaw: a python package for analysis of heavy-tailed distributions. *PloS one*, 9(1):e85777.

Yu Bai, Fan Chen, Huan Wang, Caiming Xiong, and Song Mei. 2023. Transformers as statisticians: Provable in-context learning with in-context algorithm selection. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Alberto Bietti, Vivien Cabannes, Diane Bouchacourt, Herve Jegou, and Leon Bottou. 2023. Birth of a transformer: A memory viewpoint. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901.

Zheng Chu, Jingchang Chen, Qianglong Chen, Weijiang Yu, Tao He, Haotian Wang, Weihua Peng, Ming Liu, Bing Qin, and Ting Liu. 2024. Navigate through enigmatic labyrinth a survey of chain of thought reasoning: Advances, frontiers and future. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1173–1203.

Damai Dai, Yutao Sun, Li Dong, Yaru Hao, Shuming Ma, Zhifang Sui, and Furu Wei. 2023. Why can gpt learn in-context? language models secretly perform gradient descent as meta-optimizers. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 4005–4019.

Gilad Deutch, Nadav Magar, Tomer Natan, and Guy Dar. 2024. In-context learning and gradient descent revisited. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1017–1028.

Li Dong et al. 2022. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Ezra Edelman, Nikolaos Tsilivis, Benjamin L. Edelman, eran malach, and Surbhi Goel. 2024. The evolution of statistical induction heads: In-context learning markov chains. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. 2023. Towards revealing the mystery behind chain of thought: a theoretical perspective. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pages 70757–70798.

Hamed Firooz, Maziar Sanjabi, Adrian Englhardt, Aman Gupta, Ben Levine, Dre Olgiati, Gungor Polatkan, Iuliia Melnychuk, Karthik Ramgopal, Kirill Talanine, et al. 2025. 360brew: A decoder-only foundation model for personalized ranking and recommendation. *arXiv preprint arXiv:2501.16450*.

Shivam Garg, Dimitris Tsipras, Percy S Liang, and Gregory Valiant. 2022. What can transformers learn in-context? a case study of simple function classes. *Advances in Neural Information Processing Systems*, 35:30583–30598.

Tianyu Guo, Wei Hu, Song Mei, Huan Wang, Caiming Xiong, Silvio Savarese, and Yu Bai. 2024. How do transformers learn in-context beyond simple functions? a case study on learning with representations. In *The Twelfth International Conference on Learning Representations*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *International Conference on Learning Representations*.

Yifan Hou, Jiaoda Li, Yu Fei, Alessandro Stolfo, Wangchunshu Zhou, Guangtao Zeng, Antoine Bosselut, and Mrinmaya Sachan. 2023. Towards a mechanistic interpretation of multi-step reasoning capabilities of language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4902–4919.

Pin-Lun Hsu, Yun Dai, Vignesh Kothapalli, Qingquan Song, Shao Tang, Siyu Zhu, Steven Shimizu, Shivam Sahni, Haowen Ning, and Yanning Chen. 2024. Liger kernel: Efficient triton kernels for llm training. *arXiv preprint arXiv:2410.10989*.

Seungone Kim, Se June Joo, Doyoung Kim, Joel Jang, Seonghyeon Ye, Jamin Shin, and Minjoon Seo. 2023. The cot collection: Improving zero-shot and few-shot learning of language models via chain-of-thought fine-tuning. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626.

Yingcong Li, Muhammed Emrullah Ildiz, Dimitris Papailiopoulos, and Samet Oymak. 2023a. Transformers as algorithms: Generalization and stability in in-context learning. In *International Conference on Machine Learning*, pages 19565–19594. PMLR.

Yingcong Li, Kartik Sreenivasan, Angeliki Giannou, Dimitris Papailiopoulos, and Samet Oymak. 2023b. Dissecting chain-of-thought: Compositionality through in-context filtering and learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Manqing Liu, David R Bellamy, and Andrew L Beam. 2024. Dag-aware transformer for causal effect estimation. *arXiv preprint arXiv:2410.10044*.

William Merrill and Ashish Sabharwal. 2024. The expressive power of transformers with chain of thought. In *The Twelfth International Conference on Learning Representations*.

Eshaan Nichani, Alex Damian, and Jason D. Lee. 2024. How transformers learn causal structure with gradient descent. In *Forty-first International Conference on Machine Learning*.

Max Nye, Anders Johan Andreassen, David A Abolafia, François Chollet, and Geoffrey Irving. 2021. Show Your Work: Scratchpads for intermediate computation. *arXiv preprint arXiv:2112.00114*.

Kazusato Oko, Yujin Song, Taiji Suzuki, and Denny Wu. 2024. Pretrained transformer efficiently learns low-dimensional target functions in-context. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. 2022. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*.

Madhur Panwar, Kabir Ahuja, and Navin Goyal. 2024. In-context learning through the bayesian prism. In *The Twelfth International Conference on Learning Representations*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Akshara Prabhakar, Thomas L Griffiths, and R Thomas McCoy. 2024. Deciphering the factors influencing the efficacy of chain-of-thought: Probability, memorization, and noisy reasoning. *arXiv preprint arXiv:2407.01687*.

Ben Prystawski, Michael Y Li, and Noah D Goodman. 2023. Why think step by step? reasoning emerges from the locality of experience. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pages 70926–70947.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training.

Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Neural Information Processing Systems*.

Johannes Von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. 2023. Transformers learn in-context by gradient descent. In *International Conference on Machine Learning*, pages 35151–35174. PMLR.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. 2022. Rationale-augmented ensembles in language models. *arXiv preprint arXiv:2207.00747*.

Jason Wei, Yi Tay, Mark Davies, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. 2022. An explanation of in-context learning as implicit bayesian inference. In *International Conference on Learning Representations*.

Chenxiao Yang, Zhiyuan Li, and David Wipf. 2024. An in-context learning theoretic analysis of chain-of-thought. In *ICML 2024 Workshop on In-Context Learning*.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. 2024. SGLang: Efficient execution of structured language model programs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Yuxiang Zhou, Jiazheng Li, Yanzheng Xiang, Hanqi Yan, Lin Gui, and Yulan He. 2024. The mystery of in-context learning: A comprehensive survey on interpretation and analysis. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 14365–14378.

Peizhen Zhu and Andrew V Knyazev. 2013. Angles between subspaces and their tangents. *Journal of Numerical Mathematics*, 21(4):325–340.

A Real World CoT Datasets and *CoT-ICL Lab*

In this section, we provide additional comprehensive details about the *CoT-ICL Lab* setup and draw parallels with real world NLP CoT datasets.

A.1 Algorithm for Token Processing

Algorithm 1 formalizes the token processing via $\mathcal{H}(l, \phi)$ by utilizing the corresponding data embedding matrix $\mathbf{E}_{\texttt{data}}$ and generating a single *chain token*. Formally, given a token embedding $\mathbf{e} \in \mathbb{R}^d$, the output of the MLP $h \in \mathcal{H}(l, \phi)$ is formulated as:

$$h(\mathbf{e}) = \mathbf{W}_L \left(\phi \left(\mathbf{W}_{L-1}(\cdots \phi \left(\mathbf{W}_1 \left(\mathbf{e} \right) \right) \right) \right). \quad (9)$$

Here $\mathbf{W}_l \in \mathbb{R}^{d \times d}, \forall l \in [L]$ denote the linear layers whose width is kept constant (d) across layers, and whose entries are sampled from $\mathcal{N}(0, 1)$. The usefulness of *CoT-ICL Lab* lies in its flexibility to modify Algorithm 1. For instance:

1. Future efforts can explore non-random $\mathbf{E}_{\texttt{data}}$ and also modify Step 3 to employ more complex function classes beyond just MLPs.

2. One can also explore feature aggregation techniques (Step 5-6) when scaling to ($|\mathcal{V}| > 1024$).

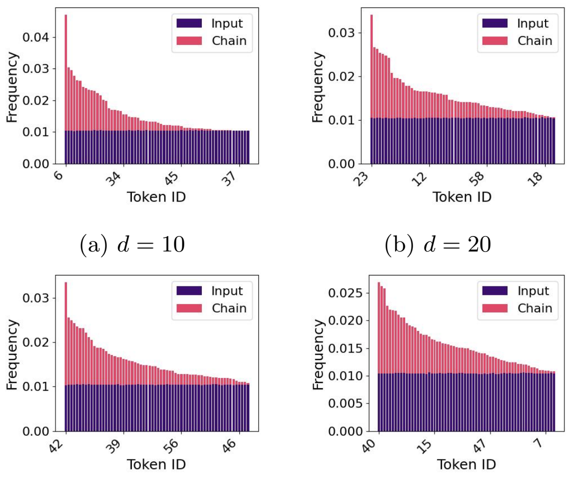
Algorithm 1 Generate a single *chain token* y_c

Require: M parent token embeddings $\{\mathbf{e}^i_{\texttt{data}}\}_{i=1}^M$ from the data embedding matrix $\mathbf{E}_{\texttt{data}} \in \mathbb{R}^{|\mathcal{V}| \times d}$, choices of depth l and activation functions ϕ,

1: Initialize MLP $h_c \in \mathcal{H}(l, \phi)$
2: **for** $i = 1$ to M **do**
3: $\quad \mathbf{h}^i \leftarrow h_c(\mathbf{e}^i_{\texttt{data}})$ {Process parent embedding}
4: **end for**
5: $\mathbf{h}_{\text{mean}} \leftarrow \frac{1}{M} \sum_{i=1}^M \mathbf{h}^i$ {Mean of features}
6: $\mathbf{h}_{\text{act}} \leftarrow \phi(\mathbf{h}_{\text{mean}})$ {Apply activation}
7: $y_c \leftarrow \texttt{argmax}(\mathbf{E}_{\texttt{data}} \mathbf{h}_{\text{act}})$ {Get chain token}
8: **return** y_c

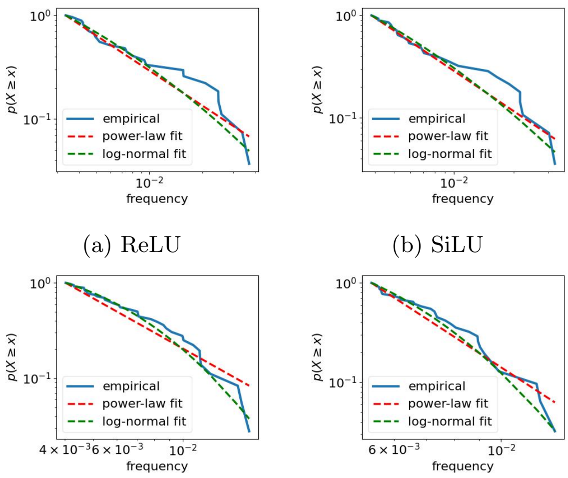
A.2 Synthetic Datasets and Token Distributions

To create synthetic training and evaluation sequence datasets via the *CoT-ICL Lab* framework, we consider a vocabulary $\mathcal{V}$ of arbitrary size and the data embedding matrix $\mathbf{E}_{\texttt{data}} \in \mathbb{R}^{|\mathcal{V}| \times d}$. To create a single sequence, we randomly sample a DAG from $\mathcal{G}(M, N, C)$ and sample C MLPs from $\mathcal{H}(l, \phi)$. The N input tokens per example are sampled uniformly from $\mathcal{V}$ and are then used to generate the C chain tokens using (2). Note that $g_c(.)$ corresponds to the M edges of the DAG that map the parent tokens to the chain token (i.e, the filtering function) and $h_c(.)$ corresponds to the token processing function via a MLP (as per Algorithm 1). Creating K such (CoT)-examples gives us a sequence and creating T such sequences gives us the desired synthetic dataset.



(a) $d = 10$ (b) $d = 20$ (c) $d = 30$ (d) $d = 40$

Figure 11: Token distribution from 10,000 CoT sequences with $|\mathcal{V}| = 64, K = 40, \mathcal{H}(1, \texttt{LeakyReLU})$ and $\mathcal{G}(M = 4, N = 4, C = 2)$.



(a) ReLU (b) SiLU (c) LeakyReLU (d) Identity

Figure 12: Complementary cumulative distribution plots of the *chain token* distribution, it's corresponding power-law and log-normal fits. We sample 10,000 CoT sequences with $|\mathcal{V}| = 64, d = 40, \mathcal{H}(1, \phi), \mathcal{G}(M = 4, N = 4, C = 2), K = 40$.

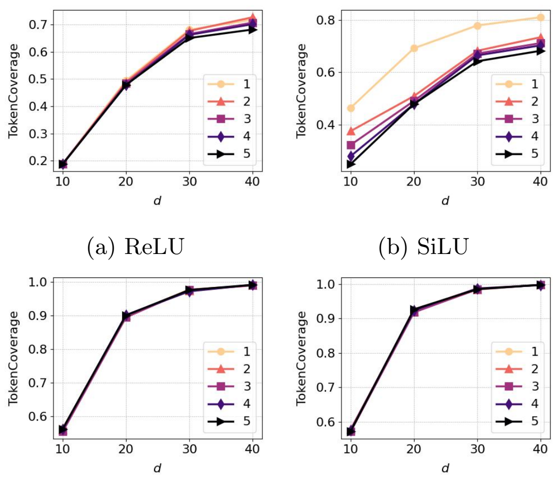
A.3 Token Distribution Fits

To understand the distribution of tokens in our synthetic datasets, we sample $10,000$ CoT sequences with $|\mathcal{V}| = 64, d = \{10, 20, 30, 40\}, \mathcal{H}(1, \texttt{LeakyReLU}), N = 4, M = 4, C = 2, K = 40$ and plot the distribution of input tokens and chain tokens in Figure 11. Observe that the distribution of chain tokens in the sequences exhibit a decay that depends on d. By utilizing the `powerlaw` python package (Alstott et al., 2014) to fit power-law and log-normal distributions to the *chain tokens* frequencies, we quantify that the distribution is more likely to be log-normal. See Figure 12 for the $d = 40$ case with varying ϕ.

To quantify the impact of varying factors (such as $\mathcal{V}, d, l, \phi$), we measure the `TokenCoverage` over all chain tokens in the dataset as follows:

Definition A.1. *The `TokenCoverage` $\in [0, 1]$ of a dataset represents the ratio of number of unique chain tokens to the number of unique tokens present in the entire dataset.*

Remark. We emphasize that `TokenCoverage` acts only as a first-order explanation of uniqueness in the entire collection of chain tokens and does not account for the unique CoT examples in the dataset.



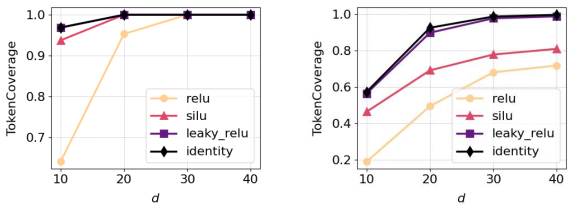
(a) ReLU (b) SiLU

(c) LeakyReLU (d) Identity

Figure 13: Measuring `TokenCoverage` of $T = 10,000$ CoT sequences with $\mathcal{G}(M = 4, N = 4, C = 2), K = 40, |\mathcal{V}| = 1024, d = \{10, 20, 30, 40\}$ and $\mathcal{H}(l, \phi)$, $l = \{1, 2, 3, 4, 5\}$ with varying ϕ.

A.4 `TokenCoverage` by MLP depth l

To illustrate the role of depth l of the MLPs in generating the chain tokens, we sample $T = 10,000$ CoT sequences with $|\mathcal{V}| = 1024, d = \{10, 20, 30, 40\}$ and $\mathcal{H}(l, \phi)$. We choose $l = \{1, 2, 3, 4, 5\}$ and $\phi = \{\texttt{ReLU, SiLU, LeakyReLU, Identity}\}$. Each CoT sequence uses $N = 4, M = 4, C = 2, K = 40$. Figure 13 illustrates that `ReLU, LeakyReLU, Identity` functions have approximately the same token coverage for varying l. However, `SiLU` tends to exhibit lower `TokenCoverage` for larger l values.



(a) $|\mathcal{V}| = 64$ (b) $|\mathcal{V}| = 1024$

Figure 14: `TokenCoverage` of $10,000$ CoT sequences with $K = 40, \mathcal{H}(1, \phi)$ and $\mathcal{G}(M = 4, N = 4, C = 2)$.

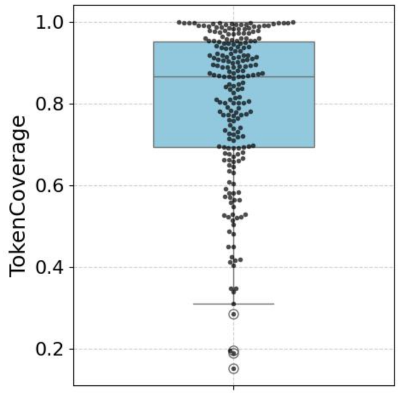


Figure 15: Measuring `TokenCoverage` of all the 216 task datasets from the CoT-Collection corpus using the Llama-3.1 Tokenizer with a vocab size of 128256.

A.5 `TokenCoverage` by MLP Activation ϕ.

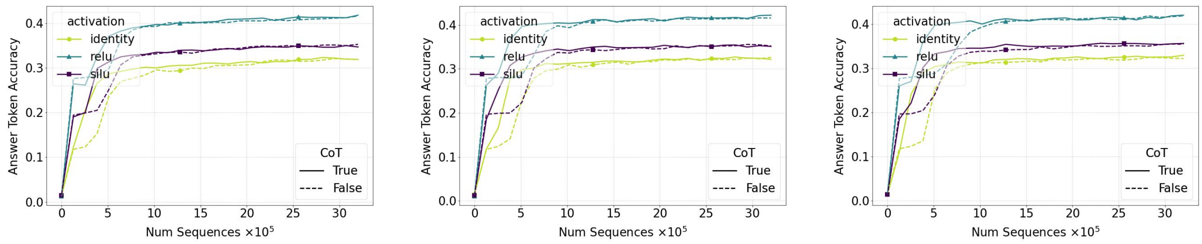
We vary $\phi = \{\texttt{ReLU, SiLU, LeakyReLU, Identity}\}$ and plot the `TokenCoverage` for a similar dataset with $T = 10,000$ CoT sequences in Figure 14. A key takeaway is that `ReLU` and `SiLU` lead to highly skewed chain token distributions with relatively smaller `TokenCoverage` for any given dimension d. Nonetheless, all ϕ exhibit an increasing trend for larger values of d.

A.6 `TokenCoverage` in the CoT-Collection NLP Dataset

To further show that our setup is flexible enough to resemble realistic NLP scenarios in token distribution, we analyze CoT sequences from the CoT-Collection dataset (Kim et al., 2023) which collates

Model	Layers	Attention Heads	Hidden Size	Intermediate Size	Params
`TF-4`	4	32	2048	8192	$243,288,064$
`TF-8`	8	32	2048	8192	$486,574,080$
`TF-12`	12	32	2048	8192	$729,860,096$

Table 1: Model Card: A naming convention for the `TF` models along with their number of layers, attention heads, hidden size, and intermediate size. The parameter count excludes the Embedding layer weights.



(a) `TF-4` (b) `TF-8` (c) `TF-12`

Figure 16: `accuracy` by varying ϕ in $\mathcal{H}(1,\phi)$ with $\mathcal{G}(M=4, N=4, C=2), d=10, |\mathcal{V}|=64, K=40$.

$\approx$ 1.8 million NLP prompts from a diverse pool of 216 tasks. We analyze the `TokenCoverage` by tokenizing the reasoning text of the NLP prompts pertaining to each task with the Llama-3.1 tokenizer ($|\mathcal{V}| = 128256$). From Figure 15, notice that the tasks span a wide range of the `TokenCoverage` values with the lowest being < 0.2 and the highest being 1. From the above analysis of simulating chains via different configurations of $\mathcal{H}, d$, we can notice that our setup is flexible enough to replicate the complexity of real-world datasets (in terms of input and chain lengths) and offer flexibility of simulating even complex datasets which might not be easy to curate.

B Hardware and Hyper-Parameters for Training and Evaluation

We use the `DistributedDataParallel` APIs in PyTorch (Paszke et al., 2019) to run each training job on 4 H100 NVIDIA GPUs. Furthermore, since our modeling code leverages HuggingFace APIs, we also employ Liger-Kernels (Hsu et al., 2024) for faster training. We created 3 different models based on the Llama-3 architecture whose details are presented in Table 1. The smallest model `TF-4` has $\approx$ 240M parameters without the embedding layer and the largest model `TF-12` has $\approx$ 730M parameters. For all the training runs, we use a batch size of 64 per rank and the `AdamW` optimizer with a learning rate 5×10^{-5}. We employ the `GenerationConfig` API in the `transformers` library to greedily generate the model predictions without teacher forcing. This API is used for evaluations on checkpoints during the training runs. For larger scale and on-demand evaluations, we provide code examples to leverage vLLM (Kwon et al., 2023) and SGLang (Zheng et al., 2024) based inference.

C Additional Experiments

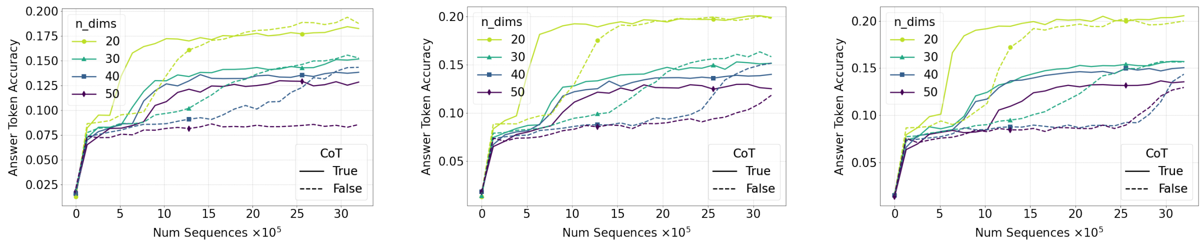
C.1 Varying Activation Functions (ϕ)

Setup. Considering a vocabulary $|\mathcal{V}| = 64, d = 10$, we employ $\mathcal{H}(1,\phi)$, $N = 4, M = 4, C = 2, K = 40$ with varying ϕ = {`ReLU, SiLU, LeakyReLU`}. All models are trained on a dataset with $T = 32 \times 10^5$ sequences and evaluated on $\widetilde{T} = 10,000$ sequences.

Lower `TokenCoverage` leads to higher `accuracy`. We previously observed from Figure 14 that the activation function (ϕ) corresponding to $\mathcal{H}$ plays a key-role in determining the `TokenCoverage` of chain tokens in the sequences. For instance, when $|\mathcal{V}| = 64$, the `TokenCoverage` with `Identity` and `SiLU` is relatively larger than `ReLU`. This implies that the number of unique tokens that a model would have to correctly predict is relatively lower in the `ReLU` case. We can observe from Figure 16 that such smaller coverage can indeed result in higher evaluation `accuracy` across all model sizes.

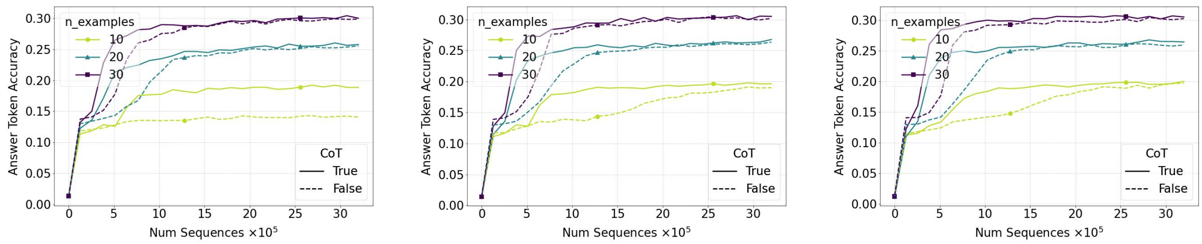
C.2 Vary Data Embedding Dimension (d)

Based on the token distribution analysis, we have noticed that the `TokenCoverage` increases monotonically with d for various choices of $\mathcal{H}(l,\phi)$ in Figure 13. As a result, Figure 17 shows that larger coverage makes it difficult for the `TF` models to attain high evaluation `accuracy` as they have to now correctly predict a larger fraction of tokens in $\mathcal{V}$.



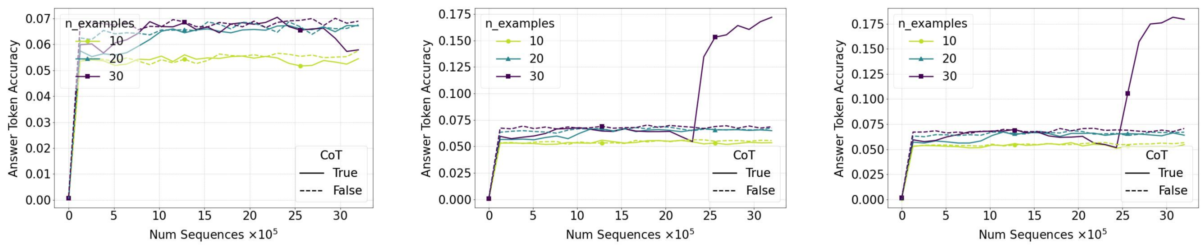
(a) `TF-4` (b) `TF-8` (c) `TF-12`

Figure 17: `accuracy` by varying d with $\mathcal{G}(M=4, N=4, C=2), \mathcal{H}(1, \texttt{LeakyRelu}), |\mathcal{V}|=64, K=40$.



(a) `TF-4` (b) `TF-8` (c) `TF-12`

Figure 18: `accuracy` by varying K with $\mathcal{G}(M=4, N=4, C=2), \mathcal{H}(1, \texttt{LeakyRelu}), d=10, |\mathcal{V}|=64$.



(a) `TF-4` (b) `TF-8` (c) `TF-12`

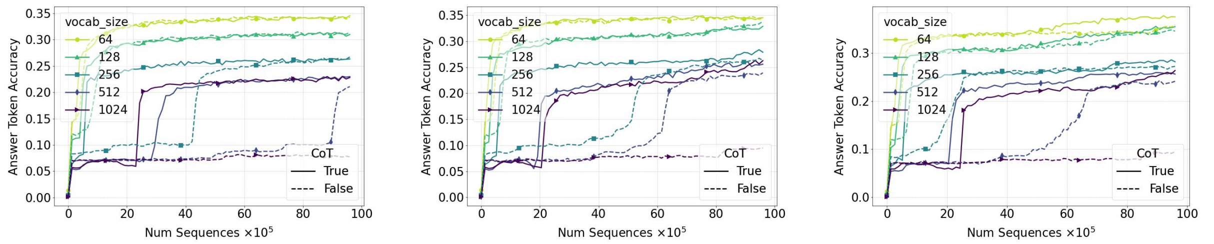
Figure 19: `accuracy` by varying K with $\mathcal{G}(M=4, N=4, C=2), \mathcal{H}(1, \texttt{LeakyRelu}), d=10, |\mathcal{V}|=1024$,.

C.3 Vary Number of (CoT-) examples K

In Section 6.1, we observed that more examples in-context can help smaller models to perform on-par with bigger models. To this end, we plot in Figure 18 the `accuracy` by varying $K = \{10, 20, 30\}$. For simplicity we choose a smaller vocab size of $|\mathcal{V}| = 64$ and show that smaller K can hurt performance even when CoT is employed across all model sizes. Additionally, larger models tend to outperform the smaller one `TF-4` in the extreme case of $K = 10$ without any CoT. Next, by increasing the vocabulary size to $|\mathcal{V}| = 1024$, we observe from Figure 19 that `TF-8`, `TF-12` leverage their depth and outperform `TF-4` by utilizing CoT when $K = 30$. For $K = 10, 20$, the problem turns out to be harder even for these bigger models.

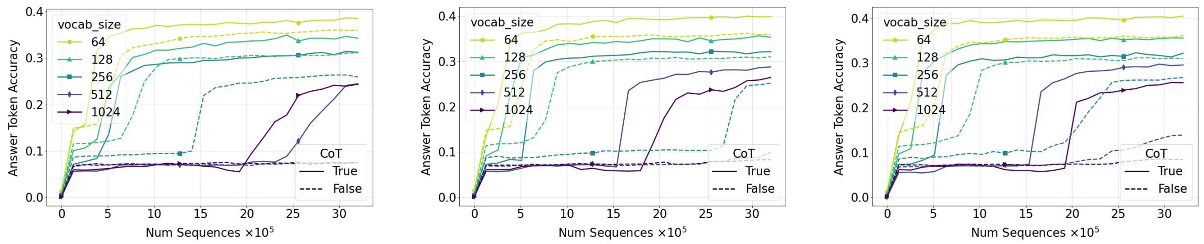
C.4 Longer Training

We follow the same setup of Section 6.1 and explore the impact of longer training on the `accuracy` of the `TF` models. In particular, we train on $3\times$ more steps, which results in 96×10^5 (CoT-) sequences per dataset. Observe from Figure 20a that by training beyond 32×10^5 sequences, even the `TF-4` model exhibits a transition in non-CoT evaluation `accuracy` for $|\mathcal{V}| = \{256, 512\}$. Similar behavior can be observed with `TF-8` (Figure 20b) and `TF-12` (Figure 20c) where the transitions tend to occur with relatively less number of training sequences. Additionally, the bigger models tend to reach a final `accuracy` with CoT which is higher than that of the smaller `TF-4` model.



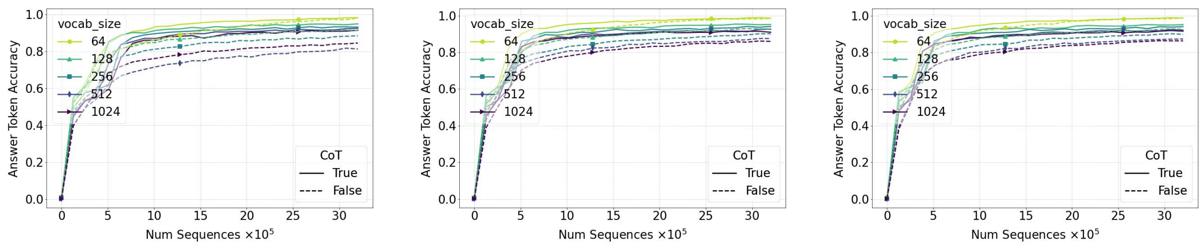
(a) `TF-4` (b) `TF-8` (c) `TF-12`

Figure 20: `accuracy` by varying $\mathcal{V}$ with $\mathcal{G}(M = 4, N = 4, C = 2), \mathcal{H}(1, \texttt{LeakyReLU}), d = 10, K = 40$ and training for $3\times$ steps (i.e 96×10^5 (CoT-) sequences).



(a) `TF-4` (b) `TF-8` (c) `TF-12`

Figure 21: `accuracy` by varying $\mathcal{V}$ with $\mathcal{H}(1, \texttt{LeakyRelu}), d = 10, K = 40$ and a **fixed DAG** sampled from $\mathcal{G}(M = 4, N = 4, C = 2)$.



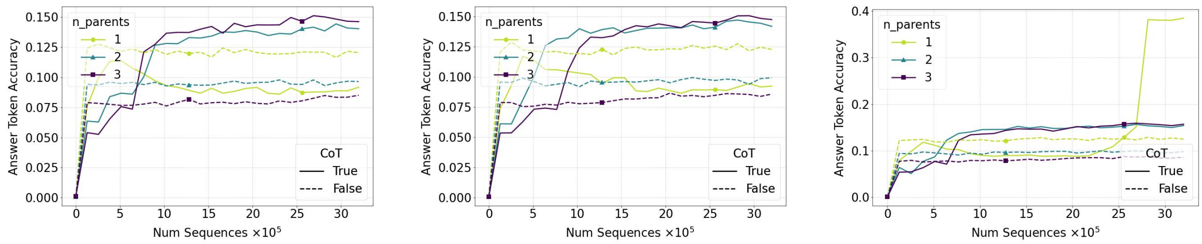
(a) `TF-4` (b) `TF-8` (c) `TF-12`

Figure 22: `accuracy` by varying $\mathcal{V}$ with $\mathcal{G}(M = 4, N = 4, C = 2), d = 10, K = 40$ and **fixed token processors** sampled from $\mathcal{H}(1, \texttt{LeakyRelu})$.

C.5 Ablations with fixed DAGs and Token Processors

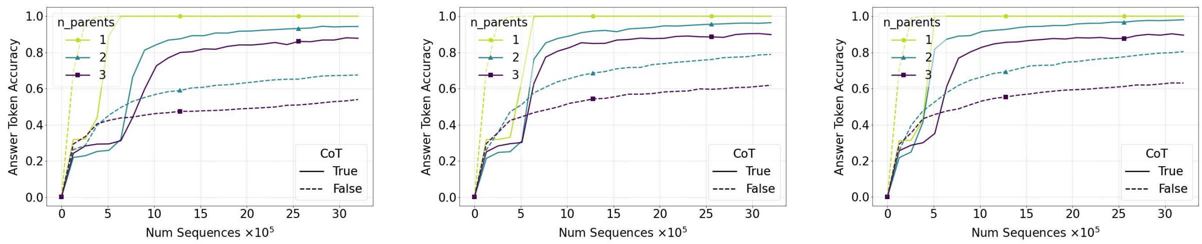
Vary $\mathcal{V}$. Following the same setup as Section 6.1 with varying vocabularies, we consider the ablations with a fixed DAG and C MLPs. We observe from Figure 21 that when the DAG is fixed across all sequences, there is a slight increase in the evaluation `accuracy` when compared to the random DAG per sequence case in Figure 3. However, notice from Figure 22 that fixing the C MLPs for all sequences facilitates the models to learn the causal structure with ease and results in very high `accuracy` when CoT is enabled. Furthermore, notice that for non-CoT datasets, `TF-8` (Figure 22b) and `TF-12` (Figure 22c) reach higher a `accuracy` than the smaller `TF-4` model (Figure 22a) for $|\mathcal{V}| = \{512, 1024\}$.

Vary M. As a follow up of Section 6.4, we show in Figure 23 that when the DAG is fixed across all sequences, the `TF-12` model exhibits a transition to higher `accuracy` (with $M = 1$) towards the end of training (Figure 23c), which is not observed in the case of `TF-4` (Figure 23a) and `TF-8` (Figure 23b). On the other hand, when we fix the C token processors, Figure 24 shows that all models can achieve a perfect `accuracy` of 1 with $M = 1$ and CoT. Interestingly, similar to the previous case of varying $\mathcal{V}$,



(a) `TF-4` (b) `TF-8` (c) `TF-12`

Figure 23: `accuracy` by varying M with $\mathcal{H}(1, \texttt{LeakyRelu})$, $d = 10, |\mathcal{V}| = 1024, K = 40$ and a **fixed DAG** sampled from $\mathcal{G}(N = 4, C = 4)$.



(a) `TF-4` (b) `TF-8` (c) `TF-12`

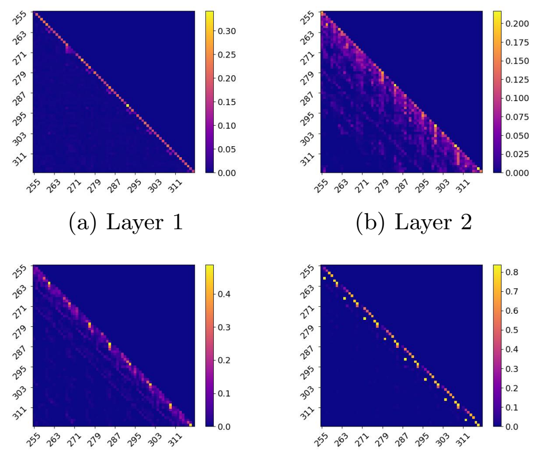
Figure 24: `accuracy` by varying M with $\mathcal{G}(N = 4, C = 4), d = 10, |\mathcal{V}| = 1024, K = 40$ and **fixed token processors** sampled from $\mathcal{H}(1, \texttt{LeakyRelu})$.

we observe that the `TF-8` (Figure 24b) and `TF-12` (Figure 24c) models reach a higher `accuracy` on the non-CoT datasets pertaining to $M = \{2, 3\}$, when compared with the smaller `TF-4` model (Figure 24a).

D Interpreting the Attention Maps

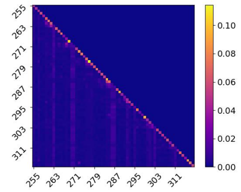
To interpret the attention maps, we consider the same ablation setup as Section 6.4 in the main text with fixed token processing functions and consider $M = 1, |\mathcal{V}| = 1024$ and the `TF-4` model for simplicity. Let the tokens x_1, x_2, x_3, x_4 and y_1, y_2, y_3, y_4 represent the input and chain tokens of a CoT example respectively. We consider a validation sequence with the DAG structure (i.e the parent tokens) for the 4 chain tokens given by: $y_1 \leftarrow \{x_4\}, y_2 \leftarrow \{x_1\}, y_3 \leftarrow \{y_1\}, y_4 \leftarrow \{y_2\}$ for the analysis. Now, given such a validation sequence prepared with $K = 40$, we autoregressively generate the 4 chain tokens and consider the attention maps used for generating the last chain token (i.e the answer token). We take the mean of all attention maps across the heads in a layer and plot the last 64 rows and columns in Figure 25. Furthermore, we also plot such attention maps for the $M = 2$ case in Figure 26 and the $M = 3$ case in Figure 27 with the DAGs described in the captions.

Qualitatively, we observed that even for the relatively difficult setup of $M = \{2, 3\}$, larger attention scores for the chain tokens in any example are placed on the parent tokens in that particular example itself (similar to the $M = 1$ case in Figure 9a). However,

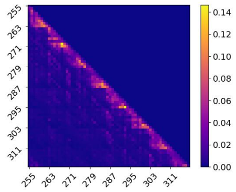


(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4

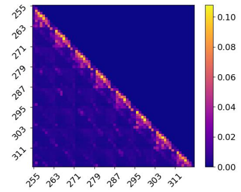
Figure 25: Mean attention matrices (last 64 rows and columns) of all 32 heads across layers of the fully trained `TF-4` model with setup: $M = 1, N = 4, C = 4, \mathcal{G}(\texttt{random})$ and fixed token processors sampled from $\mathcal{H}(1, \texttt{LeakyReLU})$. The DAG structure is $y_1 \leftarrow \{x_4\}, y_2 \leftarrow \{x_1\}, y_3 \leftarrow \{y_1\}, y_4 \leftarrow \{y_2\}$.



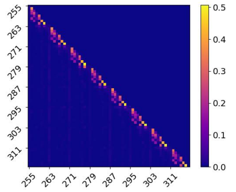
(a) Layer 1



(b) Layer 2

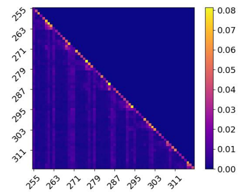


(c) Layer 3

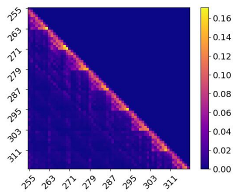


(d) Layer 4

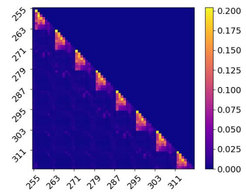
Figure 26: Mean attention matrices (last 64 rows and columns) of all 32 heads across layers of the fully trained `TF-4` model with setup: $M = 2, N = 4, C = 4, \mathcal{G}(\texttt{random})$ and fixed token processors sampled from $\mathcal{H}(1, \texttt{LeakyReLU})$. The DAG structure is $y_1 \leftarrow \{x_4, x_2\}, y_2 \leftarrow \{x_1, x_4\}, y_3 \leftarrow \{y_1, x_2\}, y_4 \leftarrow \{y_2, x_4\}$.



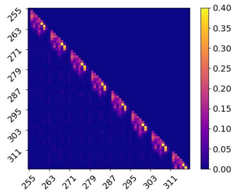
(a) Layer 1



(b) Layer 2

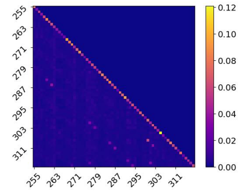


(c) Layer 3

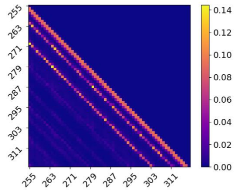


(d) Layer 4

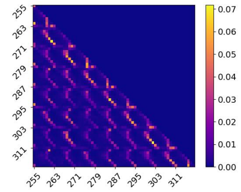
Figure 27: Mean attention matrices (last 64 rows and columns) of all 32 heads across layers of the fully trained `TF-4` model with setup: $M = 3, N = 4, C = 4, \mathcal{G}(\texttt{random})$ and fixed token processors from $\mathcal{H}(1, \texttt{LeakyReLU})$. The DAG structure is $y_1 \leftarrow \{x_4, x_2, x_3\}, y_2 \leftarrow \{x_1, x_4, y_1\}, y_3 \leftarrow \{y_1, x_2, x_4\}, y_4 \leftarrow \{y_2, x_4, x_2\}$.



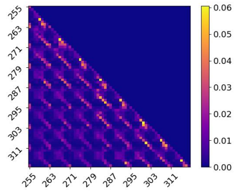
(a) Layer 1



(b) Layer 2



(c) Layer 3



(d) Layer 4

Figure 28: Mean attention matrices (last 64 rows and columns) of all 32 heads across layers of the fully trained `TF-4` model with setup: $M = 1, N = 4, C = 4, \mathcal{G}(\texttt{random}), \mathcal{H}(1, \texttt{LeakyReLU})$. The parent tokens of the chain tokens are $y_1 \leftarrow \{x_4\}, y_2 \leftarrow \{x_1\}, y_3 \leftarrow \{y_1\}, y_4 \leftarrow \{y_2\}$. When the cardinality of $\mathcal{H}$ is not bounded, we notice that the Layer 4 attention maps indicate a reliance on the answer tokens of previous in-context examples to generate final answer token.

in the generic case of randomly sampling the token processors per sequence, we observed that the mean attention scores of the last layer indicate a reliance on the answer tokens of previous examples (see Figure 28). This is an interesting observation which requires further study on the dependence between diversity of token processors and the attention map score distributions.

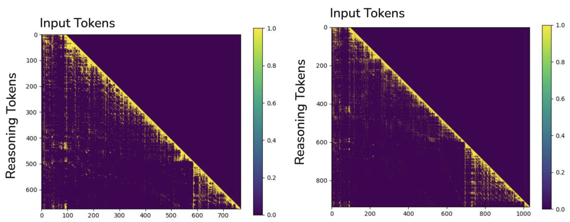
Patterns beyond the 'induction heads'. Previous works such as (Olsson et al., 2022; Edelman et al., 2024; Nichani et al., 2024) typically relied on 'attention-layer' only models to analyze the 'induction head' characteristics for improved ICL performance. Our analysis goes beyond such prefix matching and copying mechanisms of induction heads. In particular, our analysis of the attention maps and the `Precision` indicates that practical transformer models are capable of inferring the causal structure even without any prefix matching mechanisms. This indicates a deeper connection between the underlying reasoning process via chain tokens and the diversity of token processing functions that facilitate such learning of causal structures.

E On the Sparsity of Attention Maps in Reasoning LLMs

A key aspect of *CoT-ICL Lab* is the freedom it provides to control the sparsity of DAGs while creating the tokenized sequences. Drawing parallels with NLP, we aim to gain insights by analyzing the mean attention maps of the last layer of reasoning LLMs when prompted with math questions from the MMLU dataset (Hendrycks et al., 2021).

Setup. The `DeepSeek-R1-Distill-Llama-8B` reasoning model is prompted to answer a high school mathematics question from the MMLU dataset. We employ the `GenerationConfig` API in the `transformers` library to generate the response with a token limit of 1024 and various temperature values. For visualization purposes, we apply thresholding on the mean attention map using a value of 10^{-3}. The attention scores greater than the threshold are assigned a value of 1 and the rest are set to 0.

The binarized attention maps of `DeepSeek-R1-Distill-Llama-8B` can be extremely sparse. We plot the mean attention maps corresponding to the reasoning (output) tokens in Figure 29. Notice that the `temperature` not only affects the reasoning output lengths, but the attention patterns as well. Interestingly, in both the scenarios, notice that the attention scores tend to concentrate around the immediately previous tokens in the reasoning chain. The input prompt is presented in Figure 30 and reasoning outputs are presented in Figure 31 and Figure 32.



(a) `temperature=0.6` (b) `temperature=0.9`

Figure 29: Mean attention matrix of all heads for the final layer of the `DeepSeek-R1-Distill-Llama-8B` model when generating the reasoning output for a high school mathematics question from MMLU.

F Comparison with Related Work

ICL with real-valued examples. Analyzing the ICL capabilities of transformer models with synthetic data has gained massive attention in recent years. In particular, the notion of using these models as "statisticians" which can learn and approximate arbitrary function classes on real valued inputs has been widely explored (Garg et al., 2022; Bai et al., 2023; Von Oswald et al., 2023; Ahn et al., 2023; Li et al., 2023a; Oko et al., 2024) (see also (Dong et al., 2022)). Unlike the setup considered in this paper, these works consider single input-output examples $(x_k, y_k)_{k=1}^K$ in context where $x_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d) \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. The goal here is to learn the linear/non-linear functions $f \in \mathcal{F}$ such that $y_k = f(x_k), \forall k \in [K]$. Although such a setup has resulted in valuable insights on how transformers learn complex function classes (linear/noisy/sparse regression, single-index models, shallow neural networks etc), it is unclear if it is suitable to explain the ICL capabilities of transformers in NLP settings with tokenized multi input-output examples.

ICL with CoT. In the real-valued setting, a recent work by (Li et al., 2023b) explored the role of CoT for learning the MLP function classes. More importantly, they evaluate the CoT outputs of the transformers without teacher-forcing and show how this decomposition can facilitate a hierarchical layer-wise learning of the MLPs. On the other hand, several works have explored simple tokenized settings which employ markov chain like causal structures to study the attention maps and training dynamics of shallow transformers (Edelman et al., 2024; Nichani et al., 2024; Akyürek et al., 2024; Bietti et al., 2023). The main idea here is to consider a small vocabulary $\mathcal{V}$ and associate a causal dependency between tokens using single/multiple markov chains. In particular, a single input token $x_1 \in \mathcal{V}$ is used to generate the chain tokens $y_1, \cdots, y_C \in \mathcal{V}^C$ as per the transition probabilities of the chain.

We highlight that the generic setup of *CoT-ICL Lab* can specialize to a markov chain like structures (via DAGs) and also allow researchers to explore token processing functions beyond MLPs (see also Table 2) to extend such lines of work. More importantly, the multi input-output ICL setup along with data embeddings allows us to take a step towards modeling NLP like scenarios with interpretable synthetic data.

```
The following is a multiple choice question about high school mathematics.

The length of a rectangle is twice its width. Given the length of the diagonal is $5\sqrt{5}$,
find the area of the rectangle.
A. 2500
B. 2
C. 50
D. 25
Which of these options is correct? Please provide a concise explanation and either option A,B,C
or D in the format: ### Final Option:
```

Figure 30: The input prompt to the `DeepSeek-R1-Distill-Llama-8B` model.

```
Okay, I'm trying to solve this problem about the rectangle. Let's see. The question says the length is twice its width,
and the diagonal is 5\sqrt{5}. We need to find the area of the rectangle. Hmm, okay, I remember that in a rectangle,
the diagonal forms a right triangle with the length and width. So maybe I can use the Pythagorean theorem here.

Let me denote the width as 'w'. Then the length, since it's twice the width, would be '2w'. Right, so length is 2w,
width is w. Now, the diagonal of the rectangle is the hypotenuse of this right-angled triangle. The formula for the
diagonal (d) would be sqrt( (length)^2 + (width)^2 ). Wait, that's the Pythagorean theorem.

Given that the diagonal is 5\sqrt{5}, I can set up the equation: sqrt( (2w)^2 + w^2 ) = 5\sqrt{5}. Let me square
both sides to eliminate the square root. That gives me (2w)^2 + w^2 = (5\sqrt{5})^2.

Calculating each term: (2w)^2 is 4w^2, and w^2 is just w^2. So adding them together, 4w^2 + w^2 is 5w^2.
On the right side, (5\sqrt{5})^2 is 25 * 5, which is 125.

So now I have 5w^2 = 125. To find w^2, I can divide both sides by 5. That gives w^2 = 25. Taking the square root
of both sides, w = 5. Since width can't be negative, we take the positive value.

So the width is 5 units. Then the length, which is twice that, is 2 * 5 = 10 units. Now, the area of the rectangle is
length multiplied by width, so that's 10 * 5 = 50.

Looking at the options, option C is 50. So I think the correct answer is C. Let me just double-check my steps. I set up
the variables correctly, used the Pythagorean theorem, squared both sides properly, solved for w, and then calculated
the area. Everything seems to add up. Yeah, I'm confident it's C.
</think>

The problem involves finding the area of a rectangle where the length is twice the width and the diagonal is
\(5\sqrt{5}\). By denoting the width as \(w\), the length becomes \(2w\).
Using the Pythagorean theorem for the diagonal:

\[
\sqrt{(2w)^2 + w^2} = 5\sqrt{5}
\]

Squaring both sides:

\[
4w^2 + w^2 = 125 \Rightarrow 5w^2 = 125 \Rightarrow w^2 = 25 \Rightarrow w = 5
\]

The width is 5, so the length is \(2 \times 5 = 10\). The area is:

\[
5 \times 10 = 50
\]

Thus, the correct answer is **C. 50**.

### Final Option: C
```

Figure 31: The output of the `DeepSeek-R1-Distill-Llama-8B` model with `temperature=0.6`.

```
Okay, so I have this problem here about a rectangle. The problem says that the length is twice its width. Hmm,
I remember that in a rectangle, the sides are pairs of equal lengths. So if the length is twice the width,
I can represent them as variables. Let me think, maybe width is w and length would be 2w.

Then the problem mentions the diagonal is 5 times the square root of 5. That's 5\sqrt{5}. I know that in a rectangle,
the diagonal forms a right triangle with the length and width. So I can use the Pythagorean theorem here.
The theorem states that in a right-angled triangle, the square of the hypotenuse (which is the diagonal here)
is equal to the sum of the squares of the other two sides.

Okay, so the diagonal is 5\sqrt{5}, so I should square that. Let me calculate that: (5\sqrt{5})^2. Squaring 5 gives 25,
and squaring \sqrt{5} gives 5. So 25 times 5 is 125. So the diagonal squared is 125.

Now, the length is 2w, and the width is w. Plugging into the Pythagorean theorem: (2w)^2 + w^2 = (5\sqrt{5})^2.
Let me compute (2w)^2, which is 4w^2, and w^2 is w^2. Adding them together, it's 5w^2. So 5w^2 = 125.

To find w^2, I can divide both sides by 5. That gives me w^2 = 25. So w is the square root of 25, which is 5.
Therefore, the width is 5 units.

Then the length is twice that, so 2 times 5 is 10. So the length is 10 and the width is 5. To find the area of the
rectangle, I multiply length by width. So 10 times 5 is 50. Hmm, that's 50. Let me check the options. Option C is 50.
So I think the answer is C.

Wait, let me double-check because sometimes when I do math quickly, I might makes mistakes. So let me recast it.

Given length is twice the width, so l=2w. The diagonal squared equals l squared plus w squared. So substituting,
(2w)^2 + w^2 = (5\sqrt{5})^2.

That's 4w^2 + w^2 = 25 * 5. So 5w^2 = 125. Dividing both sides by 5 gives w^2=25. So w=5, since width can't be negative.
So length is 10. Area is l times w, so 10*5=50. Yep, that seems correct.
So the area should be 50, which is option C.
</think>

The length of the rectangle is twice its width. Using the Pythagorean theorem for the diagonal, we find the width and
length, then calculate the area.

### Step-by-Step Explanation:

1. **Define Variables:**
   - Let the width be \( w \).
   - Then, the length is \( 2w \).

2. **Apply Pythagorean Theorem:**
   - The diagonal is \( 5\sqrt{5} \).
   - \( (length)^2 + (width)^2 = (diagonal)^2 \)
   - Substitute \( length = 2w \) and \( width = w \):
     \[
     (2w)^2 + w^2 = (5\sqrt{5})^2
     \]
   - Simplify:
     \[
     4w^2 + w^2 = 25 \times 5
     \]
     \[
     5w^2 = 125
     \]

3. **Solve for \( w^2 \):**
   - \( w^2 = \frac{125}{5} = 25 \)
   - \( w = \sqrt{25} = 5 \) (since width can't be negative).

4. **Find Length and Area:**
   - Length \( = 2w = 2 \times 5 = 10 \).
   - Area \( = length \times width = 10 \times 5 = 50 \).

### Final Answer:
The area of the rectangle is **50**, which corresponds to option **C**.

### Final Option:
C.
```

Figure 32: The output of the `DeepSeek-R1-Distill-Llama-8B` model with `temperature=0.9`.

Previous Work	Tokenized Setup	CoT in ICL	Multi-Input Output ICL	Explicit DAG
Von Oswald et al. (2023)	×	×	×	×
Ahn et al. (2023)	×	×	×	×
Garg et al. (2022)	×	×	×	×
Bai et al. (2023)	×	×	×	×
Li et al. (2023a)	×	×	×	×
Guo et al. (2024)	×	×	×	×
Panwar et al. (2024)	×	×	×	×
Oko et al. (2024)	×	×	×	×
Xie et al. (2022)	✓	×	×	×
Akyürek et al. (2024)	✓	×	×	×
Dai et al. (2023)	✓	×	×	×
Deutch et al. (2024)	✓	×	×	×
Edelman et al. (2024)	✓	×	×	✓
Nichani et al. (2024)	✓	×	×	✓
Feng et al. (2023)	✓	✓	×	×
Prystawski et al. (2023)	✓	✓	×	✓
Merrill and Sabharwal (2024)	✓	✓	×	×
Hou et al. (2023)	✓	✓	×	✓
Li et al. (2023b)	×	✓	×	×
CoT-ICL Lab	✓	✓	✓	✓

Table 2: Comparison with related work on ICL and CoT.

Notation	Description
$\mathcal{V}$	A custom vocabulary used to generate tokens
d	The data embedding dimension in $\mathbf{E}_{\texttt{data}} \in \mathbb{R}^{\lvert\mathcal{V}\rvert \times d}$.
$\mathbf{E}_{\texttt{data}}$	A constant 'unknown' data embedding matrix
$\mathcal{G}$	A class of functions to filter/select tokens
$\mathcal{H}$	A class of functions to process token embeddings (rows in $\mathbf{E}_{\texttt{data}}$)
$\mathcal{F}$	A function class composed of $\mathcal{G}, \mathcal{H}$
l	The depth of the MLP in $\mathcal{H}$
ϕ	The activation function used in the MLP in $\mathcal{H}$
N	Number of input tokens per example
M	Number of tokens selected by $\mathcal{G}$
C	Number of chain tokens (Chain length)
K	Number of examples per sequence
$\texttt{TF}$	A decoder only transformer model
$\mathbf{x} = (x_1, \cdots, x_N) \in \mathcal{V}^N$	Input tokens in an example
$\mathbf{y} = (y_1, \cdots, y_C) \in \mathcal{V}^C$	Chain tokens in an example
$\mathbf{p}^K(f)$	A sequence composed using $f \in \mathcal{F}$ with K examples in-context.
$\mathbf{p}^K_{CoT}(f)$	A CoT sequence composed using $f \in \mathcal{F}$ with K examples in-context.
$\texttt{TF}^{\circ C}(\cdot)$	The C-step auto-regressive greedy token generation by the $\texttt{TF}$ model.

Table 3: A summary of notations used throughout the paper.